

Migrating Stovepipe Systems to Integrated/Interoperable Platforms Using the Technical Reference Model and Object-Oriented Operational Architectures

Objective of Case Study

This case study demonstrates how both the Technical Reference Model (TRM) and an object-oriented operational view can be used to migrate disparate, stove-piped systems into an integrated and interoperable system using the ideas and concepts of spiral development, evolutionary acquisition and the DoD Architecture Framework ([Figure 8-1](#)). The traditional approach to system development, testing, and deployment is to build standalone systems capable of supporting specific functionality within a given mission (i.e., space, air, missile, etc.). This design strategy laid the foundation for our current redundant system architecture, which supports many operational systems and uses different programming languages. Maintaining hardware and software upgrades on standalone architecture designs of this nature imposes major limitations. These inflexible, “stove-piped” systems cannot meet the growing information exchange requirements of today’s operational environment or provide a means to keep pace with evolving technology. The Combatant Commanders Integrated Command and Control System (CCIC2S) [previously known as the North American Aerospace Defense Command (NORAD)/United States Space Command (USSPACECOM) Warfighting Support System (N/UWSS)] project was initiated to resolving this problem of multiple, complex systems that retain an abundance of overlapping features and functions.

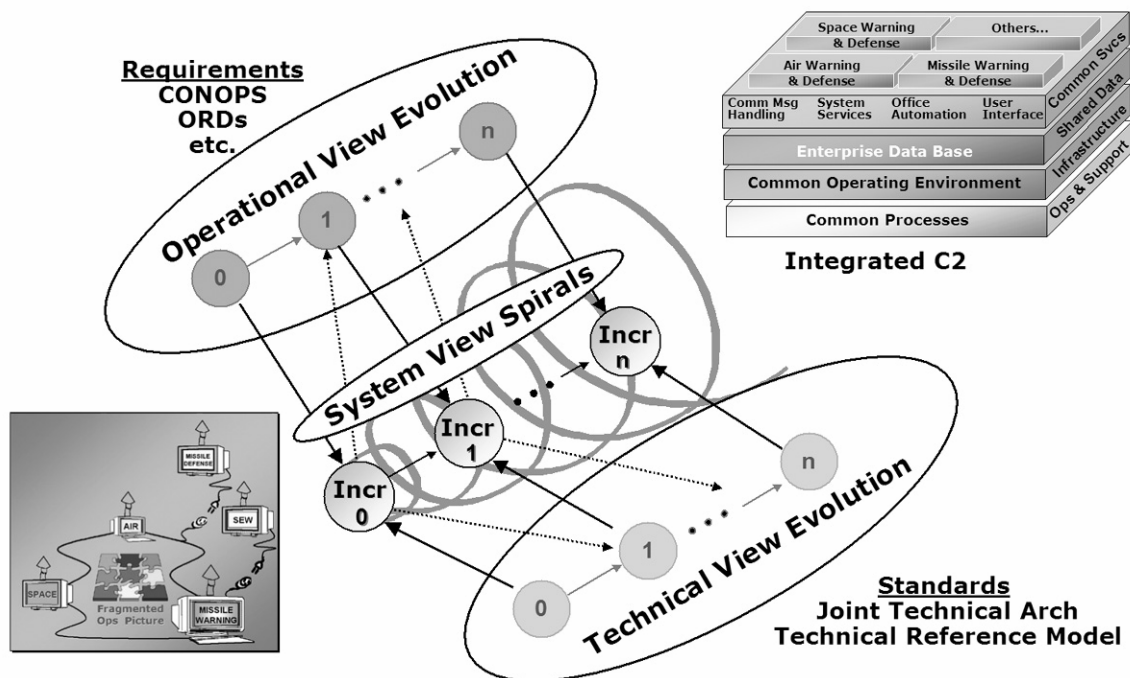


Figure 8-1. Spirally Evolving to Integrated/Interoperable Command and Control

The initial stages of the CCIC2S program were focused on identifying the baseline operational environment, determining redundancies, and describing a vision for migrating current systems. The dilemma was in determining a method to examine the operational activities across mission areas while realizing or determining functional redundancies within each stove-piped process or activity. Utilizing subject matter expertise from within the Department of Defense (DoD) community and an object-oriented (OO) methodology, the CCIC2S Core Team identified and captured system redundancies and common functions within the existing Cheyenne Mountain Operations Center (CMOC) architecture, as well as the unique functions required to perform each mission.

The migration concept describes the vision and philosophy for migrating from the current complex of systems to a single multi-layered interoperable system that enables warfighters to accomplish their mission. The vision is a virtual environment that combines access to all air, space, missile, and intelligence mission information with automatic sharing of information with any authorized user who needs it worldwide.

Using the DoD TRM and a Unified Modeling Language (UML) based operational and system views, the CCIC2S Team has created an overall Department of Defense Architecture Framework (DoDAF) model that provides the migration of legacy systems to interoperability and provides the maximum level of reusability ([Figure 8-2](#)).

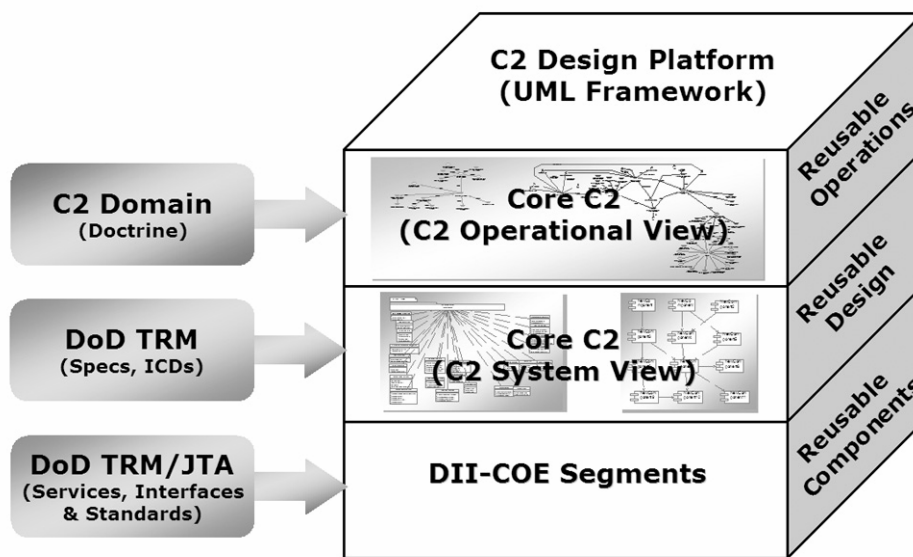


Figure 8-2. Reusable Structure of the CCIC2S Architecture Approach

Operational View

The CCIC2S Operational Architecture (OA) Team used object-oriented UML to create an OA because of its robustness in symbol sets and OO characteristics such as generalization, specialization, and inheritance. UML *use cases*¹ may modify (inherit) behavior of a second *use case*; capture data interaction among operators, nodes, and systems; and allocate behavior responsibility to systems (UML objects) ([Figure 8-3](#)).

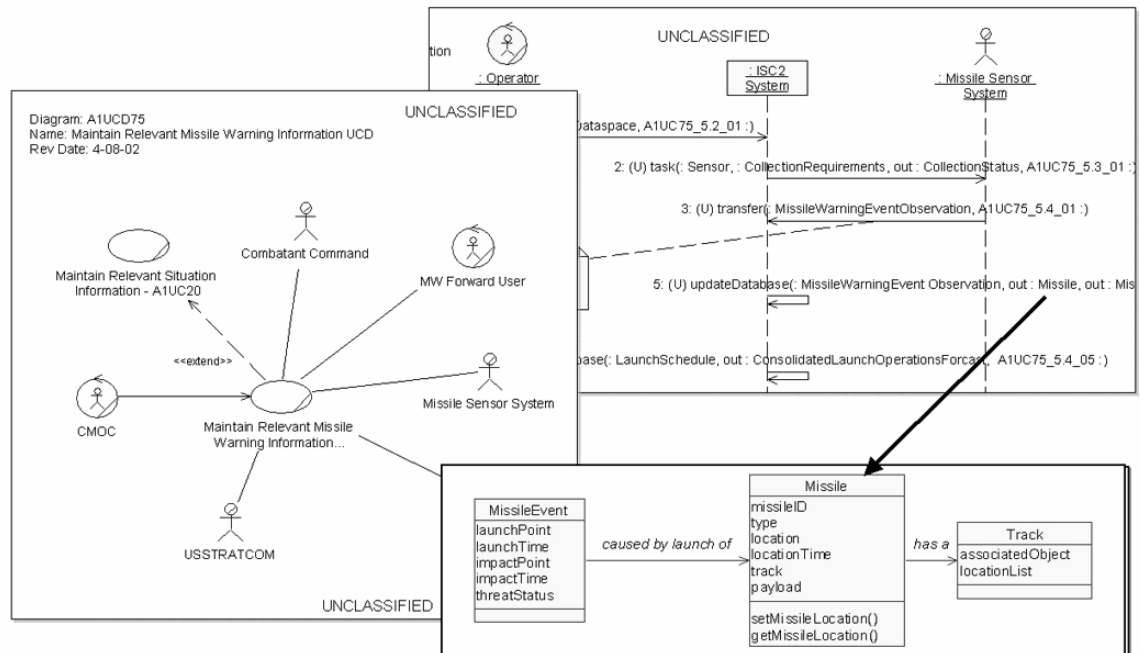


Figure 8-3. Understanding and Communicating Requirements


The UML operational architecture approach provides a comprehensive understanding of operational requirements, identifies testing and training requirements earlier in system evolution, determines visually recognizable reuse, works with smaller components, provides an open design space, and focuses on system interfaces. The UML approach has a very high focus on concepts of operations (CONOPS) early in architecture evolution and the visual aspect of *use cases* provide a standardized method to evolve the system requirements. In addition, operators, subject matter experts, and stakeholders quickly grasp the UML *use case* concept. As a result, UML provides a higher level of operator understanding of operational needs by identifying *use case* observable results of value; scope; operator, element, center, organization, and system roles; and actor specification of a sequence of actions in developing the overall enterprise. Because *use cases* are highly focused on CONOPS as they evolve to requirements, testing and training planning can occur earlier in the process. Traditionally, testing and training plans are not assembled until the system reaches a maturity level near completion. Because of the nature of *use cases*, the operational analyst can understand the relationships between *use case* results of value and more easily identify design reuse in the operational process. The process also lays the foundation for developed components that are smaller and more reusable reducing the cost of potential rework. Using the OO approach to OA, the process lays the foundation for system design without imposing technological restrictions on the developer's solution. Finally, the UML process is

1. **Use case**—A description of system behavior, in terms of sequences of actions. A use case should yield an observable result of value to an actor. A use case contains all alternate flows of events related to producing the “observable result of value.” More formally, a use case defines a set of use-case instances or scenarios. The specification of a sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system.

highly focused on system interfaces. By focusing on system interfaces, developers can produce UML system views that show product line interaction and traditional *use case* views to design and build the system. The follow sections provide a general overview of how this business-reengineering¹ concept presents operational views useful in the development of system views.

The Operational Architecture Process

The operational architecture process begins by identifying relevant *use cases* with observable results of value (ROV) distinctiveness (e.g., [Figure 8-3](#) depicts Missile Warning Information ~ ROV). Identified in the scope of the *use case*, the results of value are usually data objects (the beginning of Logical Data Model ~ OV-7 development) created or maintained by the *use case* activity. Relationships between *use cases*, if required (e.g., <<extend>>² or <<include>>³), are determined by understanding whether the extended *use case* is a modification (adaptation) of behavior⁴ of the parent case (base use case) [generally referred to as generalization/specialization⁵] or a reusable *use case* by the base *use case*. The general way to understand <<extend>> or <<include>> is that we use <<extend>> to “do on a condition of a parent case” and <<include>> “to always use a particular *use case*.” This is the foundation of use case relations in an operational level UML model and provides much payback in identifying operational patterns and reusability—essential to increase efficiency of the development activity. Ultimately, through an iterative process, the *use case* results in a System Operational Sequence (SOS) assigning behavior responsibility to the system to be built ([Figure 8-8](#)) using the ideas and concepts of the Rational Unified Process for Systems Engineering (RUP SE).

Once the *use case* and its associated scope (to include ROV) are well understood, the architect, working with subject matter experts and operators, determine relevant actors and roles involved in the *use case* activity. UML actors can be organizations, centers, nodes, or systems inside [denoted by Figure 8-4).

-
1. Business Reengineering—To perform business engineering where the work of change includes taking a comprehensive view of the entire existing business and think through why you do what you do. You question all existing business processes and try to find completely new ways of reconstructing them to achieve radical improvements. Other names for this are business process reengineering (BPR) and process innovation.
 2. Extend—A relationship from an extension use case to a base use case, specifying how the behavior defined for the extension use case can be inserted into the behavior defined for the base use case.
 3. Include—A relationship from a base use case to an inclusion use case, specifying how the behavior defined for the inclusion use case can be inserted into the behavior defined for the base use case.
 4. Behavior—The observable effects of an operation or event, including its results.
 5. Generalization/specialization—A taxonomic relationship between a more general element and a more specific element. The more specific element is fully consistent with the more general element and contains additional information. An instance of the more specific element may be used where the more general element is allowed.

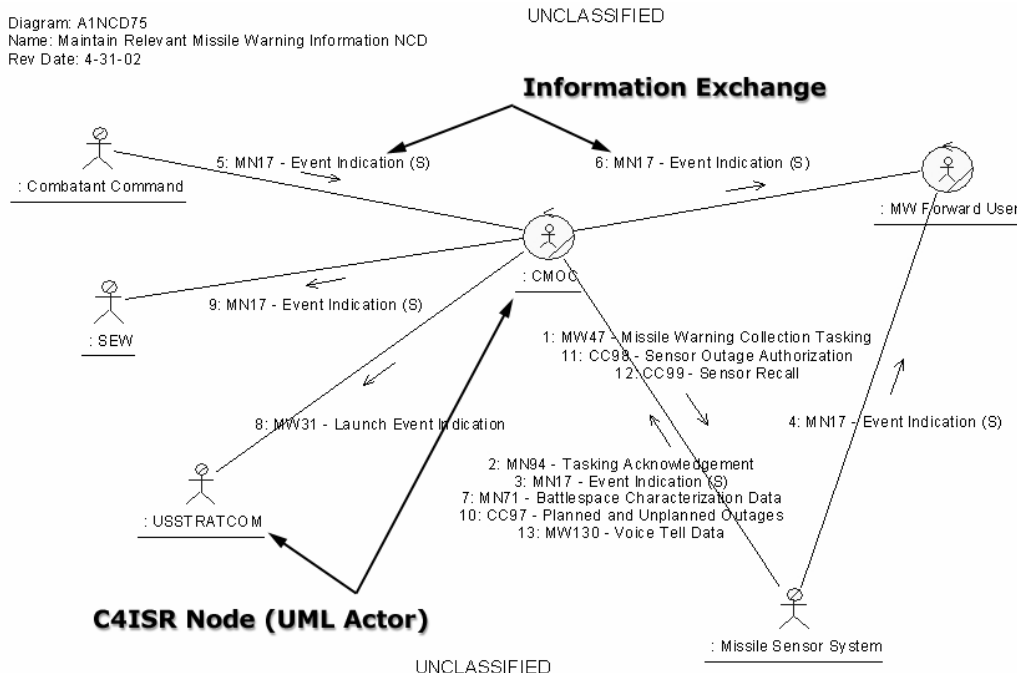


Figure 8-4. Node Connectivity Description (UML Collaboration Diagram)

Based on the evolving NCD, again iteratively, the architecture team develops the NCD Sequence diagram ([Figure 8-5](#)), a view of OV-5 and OV-2, that describes the desired activities, system transactions, behaviors between nodes and the underlying capabilities that depict the overall desired UML activity. Information exchanges are indicated by the UML message lines (lines with arrows) on the collaboration and sequence views ([Figures 4](#) and [5](#)). The collaboration and sequence views relate to each other at the node¹ and information exchange level.

The process continues until the *use case* relationship diagram (UCRD) represents the overall desired activities and their relationships (item 2 in [Figure 8-6](#)). There are other significant views such as actor relationships (refined command relationships ~ OV-4), Information Exchange Matrix (OV-3) database, and the *use case* specification which are outside the scope of this case study discussion. Together, their views and relationships are the foundation for understanding the desired operational behavior to form the operational requirements. The overall relationships between the products views are depicted in [Figures 5](#) through [11](#) showing product relationships 1 through 12 (shown by numbers in yellow circles).

1. Node—A representation of an element of architecture that produces, consumes or processes data.

Using action verb titles derived from the primary transactions developed in the NCD Sequence (Figure 8-5), the architect develops the *use case* activity diagram (OV-5) [note: one per *use case*], which show the key decision points in the operational flow and provides the view to identify consuming and producing data objects for each activity. Again, the details of this process are beyond the scope of this case study. However, because of its simplicity and visual depiction of operations, the *use case* activity diagram is a popular view with operators and stakeholders. Activity diagrams make it easier to understand the process and they show the operational flow of information necessary to support the operational activity and the underlying logical data model (~ OV-7) (Figure 8-7).

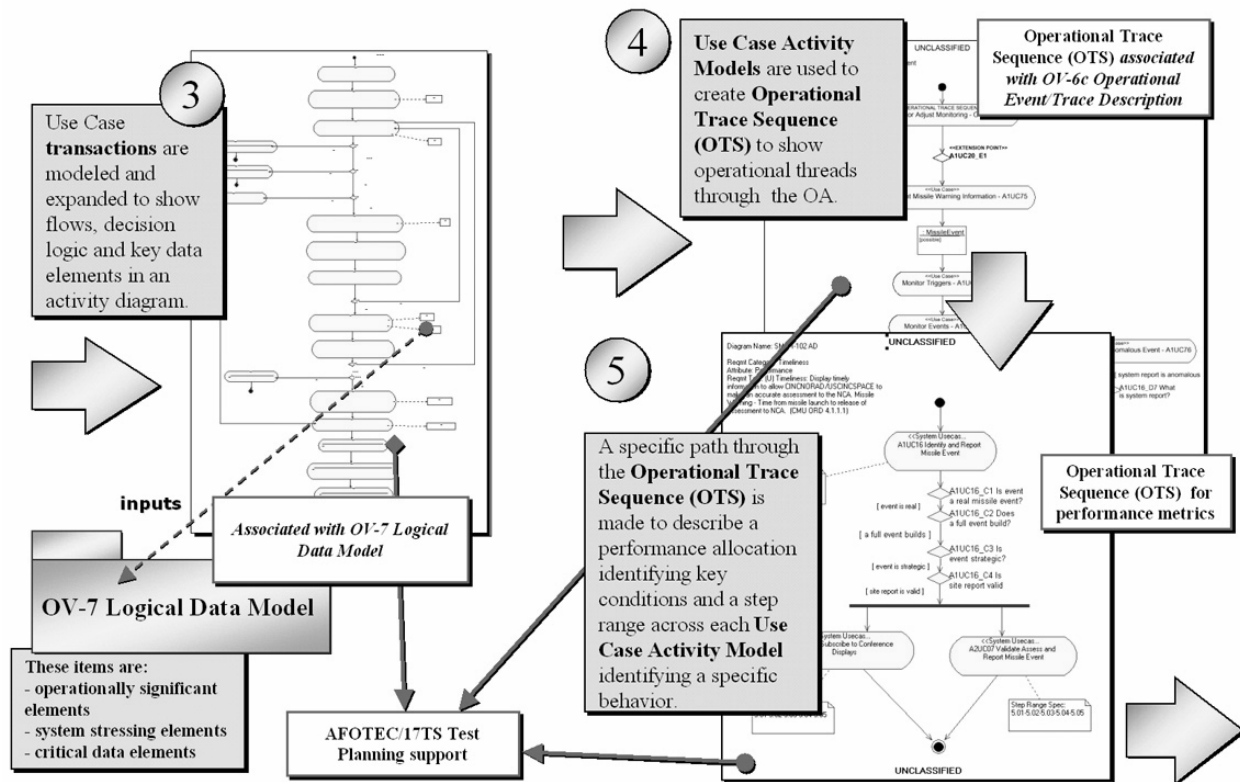


Figure 8-7. Developing the Logical Data Model (OV-7) and Operational Trace Sequences (OV-6c)

Once *use cases* mature to a conceptual level to include ROV and scope, again using UML activity models, *use cases* are abstracted to a UML activity, stereotyped as <<use case>> and form the basis for building operational threads through the model. Called Operational Trace Sequences (OTS) (~ OV-6c), these high level views are useful for representing key performance parameters (KPPs), thresholds and objectives, and for building conceptual operational threads throughout the model. Operationally significant data objects and the information they contain form the information building block to describe significant operational flow through the *use cases*. Using

the OTS, we now have a way to represent CONOPS and thread the pieces of the model in a useful way. Rational Software views our OTS as a high level scenario¹ oriented *use case*. Once we demonstrated how to represent all the necessary DoD Architecture Framework views in UML, we initially lacked a way to transition from operational views to system views. As a result, we created a new view not discussed by the DoDAF called the System Operational Sequence (SOS).

Transition from Operational View to System View

To solve the transition problem, we added an additional product to those identified by the DoDAF to join the operational and system views when using object-oriented techniques. Specifically, we developed a UML sequence diagram (Figure 8-8) [Note: one per *use case*] to explicitly allocate system responsibilities (transactions conveying data objects) to systems that satisfy the behavioral requirements identified in the operational view. The principles are the same as those discussed in the Rational Unified Process for Systems Engineering (RUP SE).

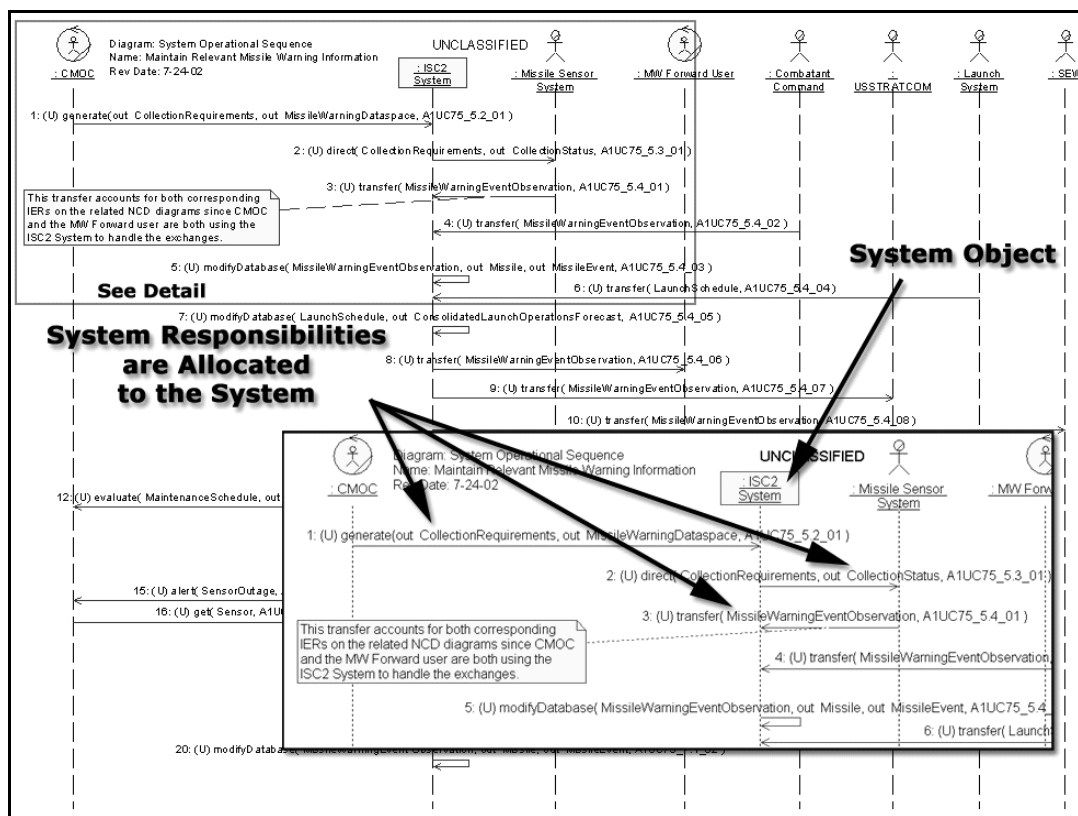


Figure 8-8. System Operational Sequence

1. Scenario—A described use-case instance, a subset of a use case. A specific sequence of actions that illustrates behaviors. A scenario may be used to illustrate an interaction or the execution of a use case instance.

This product yielded several key advantages that include:

- Provides a detailed basis for tracing operational activities to system functions, contributing the operational activities identified in the SV-5 Operational Activity to System Function Traceability Matrix of the system view.
- Allows a single operational view to support multiple systems as well as the force providers and the programs that build them.
- Facilitates the expression of lower level concepts of operation.
- Clearly identifies system boundary behavior.

Thus, we can expand the scope of an operational view beyond a single system to better define cohesive operations across a whole domain or enterprise and still explicitly allocate behavioral requirements to one or more systems.

Use cases provide insight into what operational activities the system must support and to whom the supporting system capabilities must be delivered. This provides important system to node allocation information in developing the SV-1 System Interface Description and SV-2 System Communications Description.

To better facilitate requirements management and provide a way to be tool independent, we developed a Rational Rose Script called the System Responsibility Report that pulls all the information out of the SOS and builds a comma separated view (CSV) file importable into nearly any application (e.g., Excel, Access, Oracle, etc.). The added view also allows additional traceability to logical data model elements and provides a means to perform horizontal analysis on the requirements. Finally, this additional view provides additional linkage to the system view.

Linkage between the operational and system views is also established through the logical data model (OV-7). This model identifies operationally significant objects and their relationships. The system view products show inheritance or traceability to these objects via generalization or dependency mechanisms. Therefore, they directly influence the objects in the physical data model (SV-11) in a loosely coupled manner. In addition, these objects are used on the OTS to describe the object dependency in operational threads.

OV-6c, Operational Trace Sequences are used to highlight dynamics associated with key operational threads through the architecture (e.g., demonstrate how missions are supported and indicate how performance metrics apply to operations). These are associated with SV-10c, System Event Descriptions, to support testing as capabilities are fielded. The OTS describes the required behavior of the system and the System Event Description identifies what portions of the system that provides the behavior. Using these in combination, the testers can determine what behavior to test for and what system configuration to test.

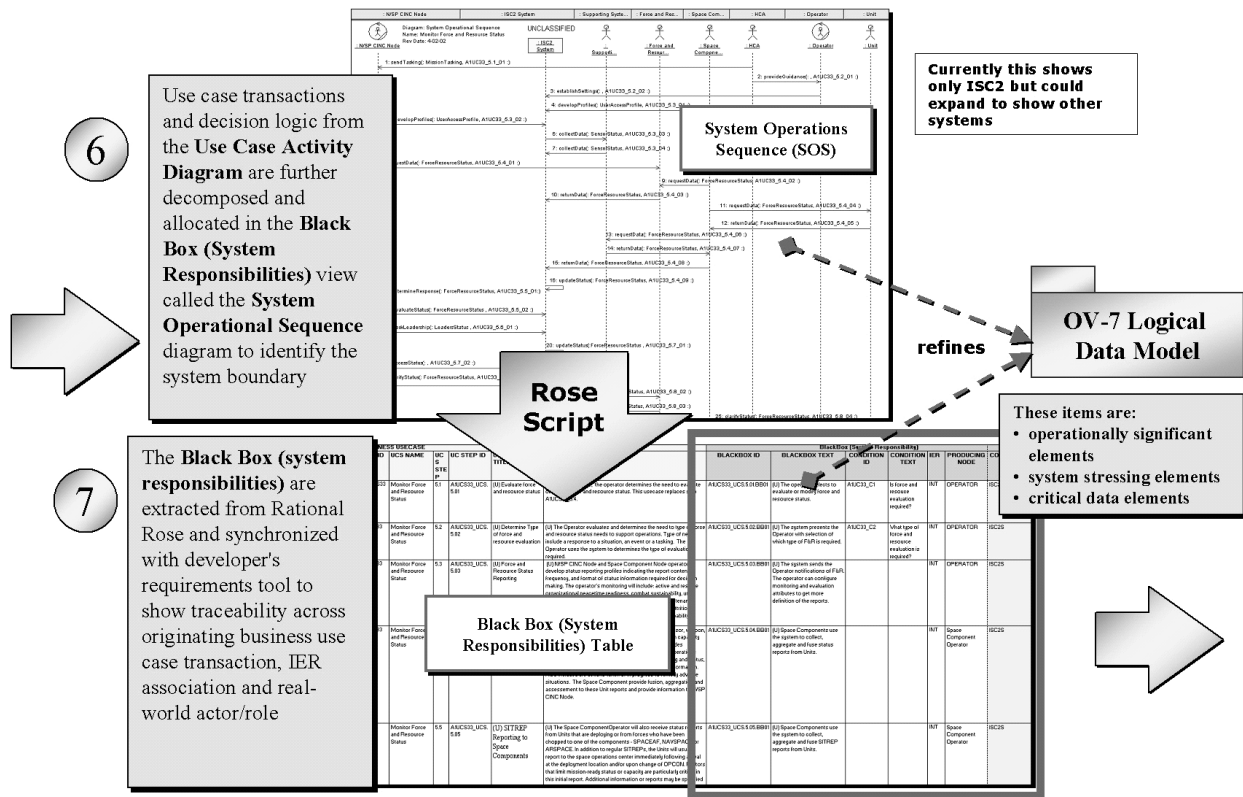


Figure 8-9. Developing System Responsibilities

System View

The system view combines the elements of the technical view to provide the behavior described in the operational view. We constructed meta-models of the various products to ensure semantic linking of the operational and system views. Although our approach can accommodate multiple systems and developers, at this time we are primarily focused on the Integrated Space Command and Control System (ISC2) contractor. We adopted a layered architecture approach in concert with TRM recommendations. The system view draws on product lines and products identified in these layers to structure the components that satisfy system responsibilities allocated to the system in the operational view ([Figure 8-10](#)). The traceability (from the OV generated system responsibilities (SOS ~ [Figure 8-8](#)) to the system components that provide functionality) is accomplished through a recursive set of sequence diagrams, allocating the responsibilities to progressively finer grain system elements from product lines through their constituent components (see “Rational Unified Process for System Engineering [RUP SE] 1.0, a Rational Software White Paper” for discussion of such an approach).

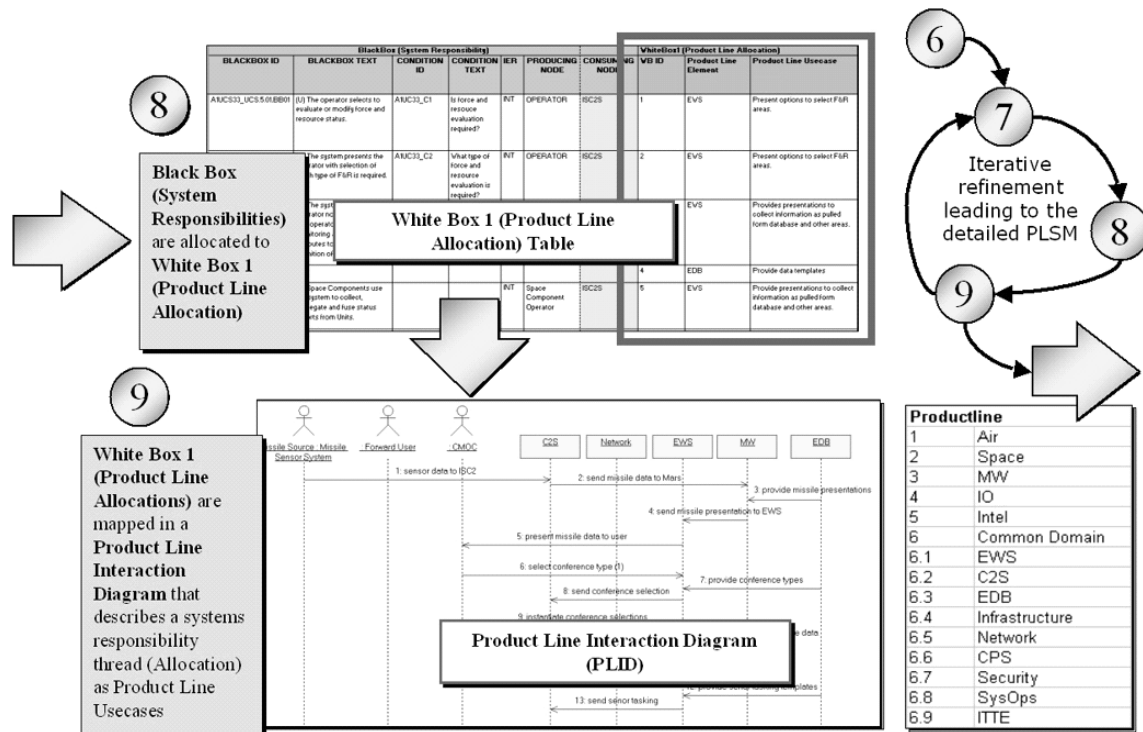


Figure 8-10. Mapping System Responsibilities to Product Lines

This process establishes the architectural and design structure that ensures that the components work together to produce the required system behavior needed by operators to conduct operations. Design activities such as modeling, coding, other generation representations (e.g., XML) and roundtrip engineering then produce the code that completes the system. Using the TRM, Specifications and Interface Control Documents are applied at this level of design. Deployment views show how the software components are fielded on hardware components as well as how the later are interconnected. This becomes a further basis for SV-1, the System Interface Description, and SV-2, the System Communications Description.

While the system view describes the “to-be” visionary architecture, we are transitioning from legacy systems to future systems that embody the architecture. We must convey how the intervening mixture of legacy and future systems are deployed and cooperate to maintain continuity of operations over the development period. This is accomplished though a series of data driven deployment diagrams based on operational delivery plans.

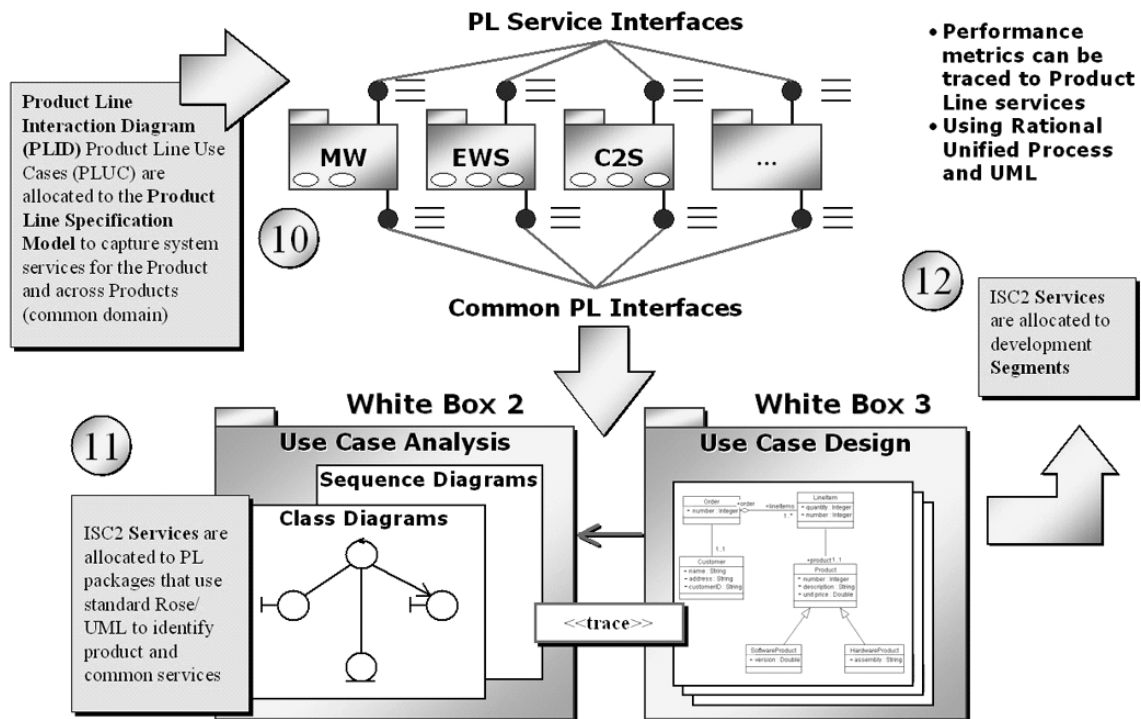


Figure 8-11. Refining Product Line Allocation to Components

Transition from System View to Technical View

The nominal interpretation of the Technical Architecture View is a minimal but sufficient time-phased list of standard technology specifications applicable to the realization of the system's requirements. Time phasing consists of delineating the current specifications, forecasting the major technology standardization trends and updating the associations between standards and system architecture elements as the system matures through its evolution. Since the standards are selected after related system capabilities are identified, it is natural to think in terms of a sequential process in which the specifications are selected after the system architecture has been defined. However, the real process, whether formal or not, involves intense analytical interaction between the system architecture and technical architecture domains. From the system architecture perspective, the design of both the logical and physical aspects is informed and constrained by what the architects consider practical in terms of available technology, standards and architectural patterns. From the perspective of the technical architecture, the selection of specifications from the vast domain of technology specifications must, in turn, be filtered by the context of the system architecture.

The success of this highly iterative interaction between views is currently quite dependent on the artfulness and experience of the architects. This is evident in light of orthogonal integration of emerging C2 standards, such as COE, with other major system architecture views. The future hope is that the framework will mature to provide readily accessible architecture patterns and technology specifications pre-organized by system domain.

The range of specifications and patterns that must be considered in this process within the domain of strategic command and control systems are currently dominated by component container middle-ware technology, tiered client-server patterns, object-relational mapped persistence, public key infrastructure, and communication technologies that cross the full spectrum of geographic distribution. In addition, technologies developed by the DISA Network-Centric Enterprise Services (NCES, formerly the DII COE) effort have recently become available for this domain. Significant examples of COE standards include workstation and user interface facilities, various types of message processing, and software build configuration management mechanisms. Further, each of these standard areas is experiencing strong change trends that must be forecast against the expected evolutionary life of the system.

Traceability from System View to Technical View

The problem of associating the domain's standards, patterns and their trends with logical and physical elements in the system architecture is a relational challenge for both system and technical architectures. The ISC2 developer addressed the problem with a few principles.

The first principle is called “greatest scope of technical constraint” in which a technical constraint should be mapped to the applicable system architecture element with the largest technical scope. The primary benefit of using the greatest scope principle is that the scope hierarchy inherent in the system architecture is leveraged to eliminate tedious, redundant, error-prone and probably unsustainable mappings between details of a standard and the recursive decomposition of the relevant system architecture element. (For example, user interface standards should be mapped to the enterprise workstation rather than each individual user interface display, or a security guard pattern should be mapped to the entire communication processing system rather than each individual guard element).

The second principle is “no orphan standards.” This principle is easy to understand but may be difficult to implement given the sheer size of strategic C2 domain. It can be a substantial effort to review the mapping to ensure that every standard listed in the technical architecture is indeed associated with and appropriately constrains some system architecture element. There are several obvious benefits: reduced architect and implementer learning load and reduced workload for quality assurance. A more subtle benefit is that building and cross checking the mapping for orphans provides an important cognitive review of the architecture.

These principles lead to a simple relational expression for the mapping as follows:

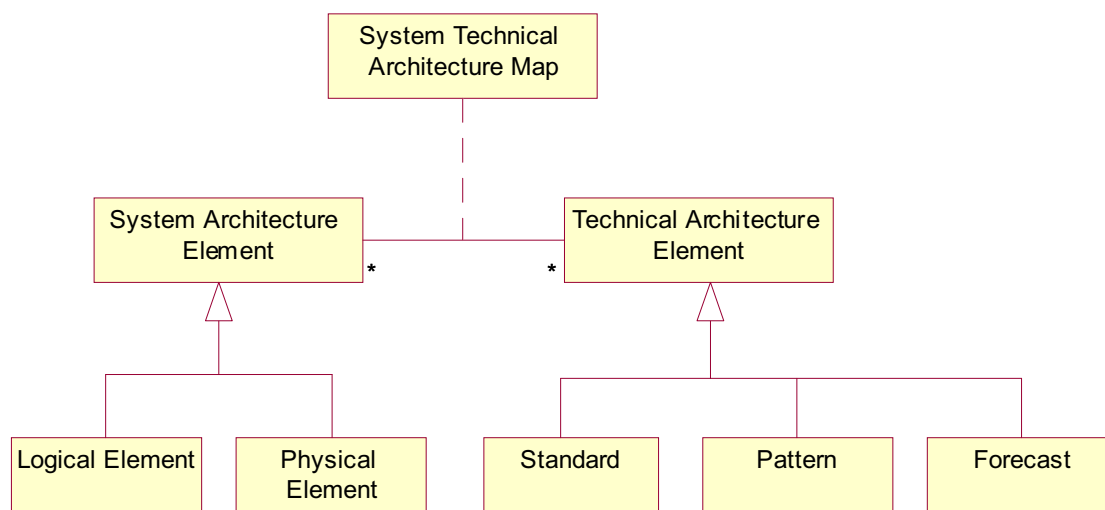


Figure 8-12. Technical Architecture Map

This pattern is easily implemented using any relational tool such as a relational database (i.e., Access, Oracle, etc.).

Technical View

In summary in terms of traceability, the operational architecture provides the source of functional requirements for the CCIC2S enterprise system. The functionality identified is based on Tier 1 C2 Battle Management and support mission functions based on traditional (existing) and non-traditional (new, emerging) threats. The CCIC2S-ISC2 requirements flow-down process allows the capability to be defined, refined, aligned and allocated in terms of functional and performance requirements to selected logical systems of interest (associated with current existing or future systems). The functionality is subject to factoring, aggregation, consolidation and realignment to core system capabilities. The allocations also include the complex cross mapping with legacy (existing) systems both those that are and are not migrating and the new/emerging systems of interest. This is to focus uniquely on mission applications while enabling the reduction/consolidation of infrastructure and common/shared functionality. The next flow down association is to the technical architecture.

The JTA provides the DoD with the fundamental building codes for the Warfighter to develop the capability for interoperability, seamless information flow and plug-and-play. The CCIC2S program assimilated the JTA, the current as is CCIC2S environment and developed a minimum set of standards, the CCIC2S TA (CSTA), which is applied to the ISC2 program. This set of mandated standards and guidelines provided the starting point for the evolution of the CCIC2S enterprise systems architecture. The ISC2 developer took this set of standards and applied in-depth industry evaluations, trade studies and comparative analysis with other standards needed to achieve the defined system functionality that was allocated to ISC2 system from the CCIC2S Operational Architecture. This evaluation continues, to keep the comparative analysis current. It provides a constant forward-looking perspective to exploit new standards and technology, a constant examination of others standards within the CCIC2S enterprise to consider in the ISC2 System Architecture, and a constant scrutiny of the need for a corresponding service within the ISC2 enterprise system. The technology forecast tracks near and long-term technology trends in order to identify promising new technologies that can be effectively applied to reduce ISC2 evolution risks and costs and increase capability. New technologies are constantly evaluated from a cost/benefit standpoint to determine applicability to future ISC2 releases to deliver the most capability for the minimum cost.

The ISC2 program falls within the NCES COE environment and also the Tier 1 and lower echelon command and control (C2) business area. Currently, the Global Command and Control System (GCCS) is emerging as the core Warfighter distributed, federated C4I system. By default, the set of capabilities with the DISA-provided COE become the starting point for capabilities mapping between the associated system-define capabilities and the DISA-provided COE capabilities. The ISC2 developer is tasked to reuse, expand, enhance, tailor or build new COE capabilities following the standard DISA processes. The ISC2 developer is exploring enhancing these processes to support faster cycles for spiral evolution, for research and development and to meet user needs for short/no-notice response mission demands. The ISC2 program open standards approach emphasizes architecture constraints and driving requirements in the selected standards and technologies, which are defined in the CSTA.

The demands of legacy system migration provide a challenge to the TV-1 and the TV-2 Technology Forecast. The ISC2 developer must manage a diverse set of standards that at times conflict or cannot be applied until a certain point in the phased migration of legacy systems. And this must be done with no impact to on-going mission operations. This complex cross phasing has been described as changing an engine while in flight. The ISC2 developer has established a full cross-matrixed ISC2 product line with capability deliveries and synchronization points that align selected systems migrations/deployments. This is managed in the ISC2 Master Integrated Evolution Plan (IMEP). This evolution involves applying the CSTA to enable a core systems infrastructure and a core database infrastructure. These areas are further permuted by other CCIC2S enterprise systems (outside the current scope of the ISC2 program) that are themselves migrating/evolving and which are within the JTA but not completely compliant with the ISC2 TA. The ISC2 developer must maintain situational awareness of all external ISC2 interfaces (functionally derived from the OA) to apply a standard industry or custom technological solution to bridge these systems. The ISC2 developer uses the TV-1 and TV-2 to inform, collaborate or guide other programs that need to interface or integrate with ISC2 core systems. This common foundation also supports systems-of-systems testing, joint testing, scalable product line, flexibility in evolution and higher fidelity in a capability-component based architecture.

Additional challenges are emerging with the DISA-provided COE and the GCCS environment in terms of standards that define specific capabilities of legacy systems that are either not needed or used by the GCCS community as a whole (due to mission uniqueness) or require state-of-the-art abilities to support real-time information exchanges or capabilities (such as for battle management execution and runtime). The ISC2 developer is also working to evolve multiple environments across various missions to a common framework while being constrained by current legacy warfighter processes and warfighting paradigms. Another challenge to a common TA application is the distributed nature of the CCIC2S environment into Warfighter environments (Combatant Command/Theaters). The ISC2 developer is focusing on instantiating mission portals, either as client/server or very normalized COTS/GOTS structures to be able to respond to Warfighter mission needs, in some instances regardless of whether the Warfighter is COE compliant or non-compliant (such as web and client/server technologies). The ISC2 TV-1 and TV-2 documents have to be dynamic “living” documents and identify elements that are sustainable and affordable.

The ISC2 developer is evolving a DISA-provided COE compliant system with standard segment taxonomy structure aligned with the CSTA. The ISC2 Product Line was also designed from its inception to have a similar taxonomy of functionality partitioning aligned with the DISA-provided COE architecture including the concepts of API layers, kernel capabilities, SHADE, COTS and GOTS, Style Guides and segmentation design. The ISC2 developer is evolving the ISC2 system to meet COE compliance in both structures, constructs and processes. The development of COE applications and components each offers various options on sustainment and processes to follow. The ISC2 Developer is evolving these options successfully to facilitate spiral evolution and enhancement of migration.

Traceability from Technical View to Technical Reference Model

The purpose of the Technical Reference Model (TRM) is to provide a common conceptual framework in a defined common vocabulary of the various components of the target system. The TRM provides the taxonomy for identifying a discrete set of conceptual layers, entities, interfaces and diagrams that provides for the specification of standards. The IMEP, Section 2 – Target System Architecture is designed to be aligned with TRM constructs. The ISC2 architecture model describes the application layers, data services, distributed operations management services, middleware services, network, platform, security, and web services. The ISC2 TSA was designed to support the DoD TRM and to provide a common vocabulary to define the ISC2 open systems services and capabilities to enable interoperability, scalability, software reuse and facilitate product line manageability. The ISC2 Product Line aligns with the CCIC2S operational architecture, COE, and TRM constructs. The ISC2 architecture solution is a standards-based implementation of an E-Business system design, leveraging mature commercial capabilities to bring robust mission capabilities to any authorized Warfighter, anywhere, at any time. It also provides a high degree of flexibility and scalability to accommodate changes in CONOPS, threats, and the resultant impacts on sensors, internal and external interfaces, mission capabilities, and users. The ISC2 net-centric model shown in [Figure 8-13](#) demonstrates how the ISC2 Product Line is aligned with the COE segmentation approach.

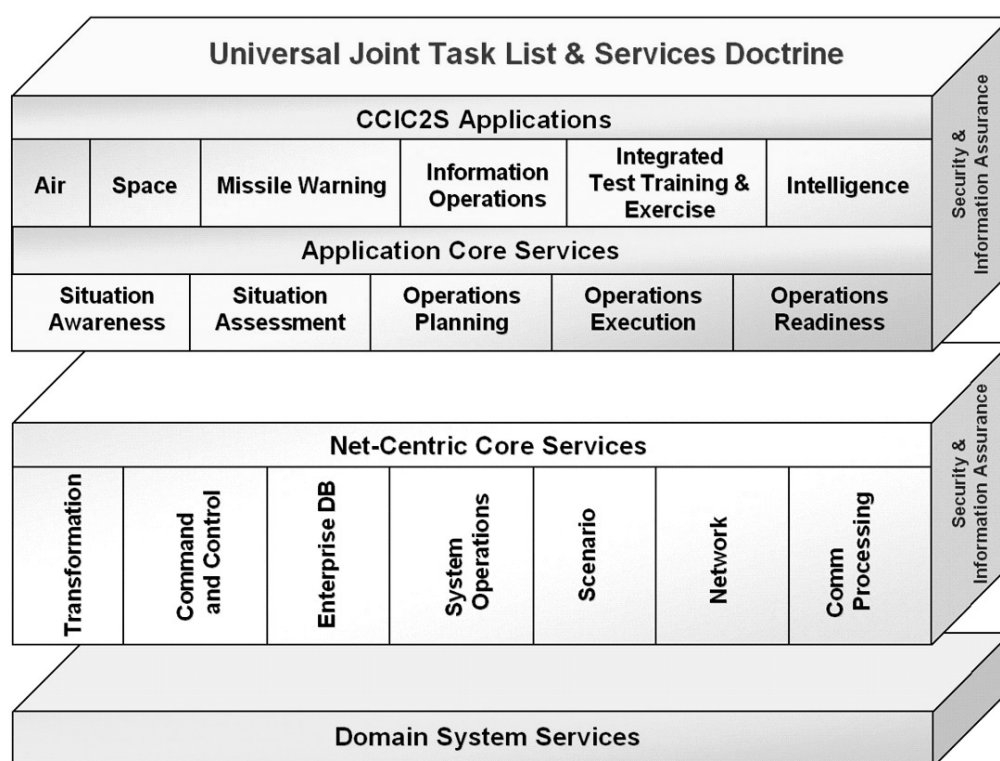


Figure 8-13. Integrated Space Command and Control Net-Centric Model

enhance the ISC2 model. This parallels previous efforts in the development of the DoD TRM that subsequently resulted in the establishment of a singular referential platform-centric TRM that is tailorable for all DoD domains. The expectation in this early stage of model maturation is that identification of and convergence to a DoD Net-Centric Reference Model will facilitate the development of Net-Centric Enterprise Services (NCES) segments and other reusable software

Our team has developed a method to achieve full DoD Architecture Framework traceability while migrating to interoperable systems using the ideas and concepts of the DoD Technical Reference Model and object-oriented UML operational and system views. We have overcome many hurdles to include end-to-end traceability and the difficult migration problems to spirally evolve stove-piped systems to an interoperable common operating picture. Using industry best practice and expertise from many leading edge companies, we have teamed to solve a very complex and difficult problem that continues to agitate developers throughout the DoD (i.e., ability to trace and link requirements across the architectural views—operational, technical and system; and integrating the DoD TRM and its methodology to support interoperability and technology insertion/transition issues). Our approach, too good to be true by many, is a seamless and systematic approach to the complex problems the DoD must face to enter the net-centric solutions in the future. According to the Clinger-Cohen Act, we must look to industry for solutions. Clinger-Cohen's vision is proving itself to be a powerful way ahead for the Department of Defense.

Acknowledgment

We would like to thank the following individuals who have contributed to this case study—they include: Rob Byrd (SI International), Tom Folk (MITRE), Dr. Paul Bailor (Lockheed Martin Mission Systems), Albert Robredo (Ciber), Mike Canaday (Master Solutions). We would also like to extend a special thanks to the members of the DoD Technical Reference Model Working Group, especially William Wong (DISA), Mike Carrio (ARTEL), and Bob Starek (Navy/METOC).

References

1. C4ISR Architecture Framework, Version 2.0, 18 December 1997
2. DoD Architecture Framework Version 1.0 (draft)
3. Rational Unified Process for Systems Engineering (RUP SE) 1.0
4. Rational Unified Process, Rational Software
5. RUP SE White Paper, Rational Software
6. Information Technology Management Reform Act of 1996
7. Technical Reference Model Version 2.0, 9 April 2001

